Recommended Documentation Format: Markdown (.md)

After conducting thorough research, I recommend using Markdown (.md) as the primary format for our project's documentation. Markdown offers simplicity, compatibility with GitHub, and flexibility, making it ideal for collaborative technical documentation.

References/Educational Resources:

GitHub Documentation: A great resource to learn more about Markdown's capabilities is available here.

Video Tutorial: This 17-minute video provides a quick introduction to Markdown for beginners.

Key Advantages of Markdown (.md):

Follow these steps to design and plan out your research. For further help and advice please speak to the course leader. Once you have filled out each section of stage 1 you are ready to start your research. Before you start your research it is also worth finding out if this research has already been done, or if you could join in with other research going on in the area.

1. **Ease of Use**

Markdown is straightforward and easy to learn. Even contributors with minimal technical experience can read and write it without difficulty.

2. **Github Compatibility**

GitHub natively supports Markdown, rendering it beautifully in repositories without any extra configuration.

3. **Tool Integration**

Markdown integrates seamlessly with Git tools, which helps track changes and diff modifications. Editors like VS Code support Markdown with real-time preview functionality.

4. **Formatting Options**

Markdown supports headers, lists, tables, images, and code blocks, making it versatile for technical documentation without being overly complex.

Contribution Guidelines

When contributing, it’s important to maintain a consistent style and structure throughout the documentation. This outlines standards for documentation could look like and how developers could contribute to the project, including updating documentation.

1. **Instructions to Include**

Sections to potentially include:

- Steps for proposing changes (issues, pull requests, etc.)
- Code standards (indentation, naming conventions)
- Documentation standards (headers, code blocks, etc.)
- How to document new features
- The review process for contributions

Example:

```
## How to Contribute


- Fork the repository.

- Create a branch for your changes.

- Document new features in the relevant section of README.md.

- Submit a pull request for review.
```

2. **Additional Documentation Files**

For larger projects, it could provide useful to split documentation into separate files, such as:

- `SETUP.md`: Instructions for setting up the project environment.

3. **Adding New Features**

Developers should document new features in the dedicated file(s. Each entry should include:

- Feature Name
- Description
- How to Use
- Related Files

Example:

```
## Feature: User Authentication

- Description: Secure user authentication with password hashing.

- Usage: Call `/login` endpoint to authenticate users.

- Related Files: `auth.js`, `login.js`
```

4. **Code Blocks**

Code blocks can be used to show examples.

Example:

```
```js
function example() {
  console.log("Code block example");
}
```

5. **Review Process**

Documentation changes should follow the same branch and pull request process as code.

6. **Consistency**

Follow a common structure and formatting style across documentation file(s).

7. **Clarity**

Keep documentation concise and straightforward.

8. **Version Control**

Update the documentation with each code change to keep it aligned with the project's current state.

9. **Conflict Resolution**

When multiple developers work on the same file, Git flags any merge conflicts, which should be resolved manually before proceeding with the merge.
```